Ernst-Abbe-Hochschule Jena
University of Applied Sciences

REIGHT 1

RISC 8-BIT CPU

B. ENG. Saitz, Ruben Hermann Felix

Ernst-Abbe-Hochschule Jena
02.01.2024

Ernst-Abbe-Hochschule Jena
University of Applied Sciences

REIGHT 1

8-bit RISC Microcontroller

with a clock speed of up to 130MHz

Datasheet

Features

- High-performance 8-bit microcontroller
- Advanced RISC architecture
  - 48 powerful instructions – almost all single-clock-cycle execution
  - 32 bytes of general-purpose working registers
  - Up to 130 MIPS throughput at 130 MHz
- High-endurance Memory
  - 1K bytes internal SRAM (1010 freely usable)
  - 512 x 16 Program Memory
- I/O
  - Inputs:
    - 5 buttons arranged in a cross shape
    - 16 switches
  - Outputs:
    - 16 LEDs
    - 4 7-Segment-Displays
  - Support for adding an I^2C interface[1]

[1] IO-Registers are implemented in design.

Table of contents

List of figures and tables

List of abbreviations

PC	Program Counter
PM	Program Memory
IF	Instruction Fetch
RF	Register File
Reg File	Register File
Mux	Multiplexer
ALU	Arithmetic-Logic-Unit
SP	Stack Pointer
DM	Data Memory
I/O	Input Output
SREG	Status Register
RAM	Random-Access-Memory
WE	Write Enable
WD	Write Disable
FF	Flip Flop
LUT	Lookup-Tables
LUTRAM	Lookup-Tables configured as RAM
BUFG	Global Clock Simple Buffer
PCB	Printed Circuit Board

1. System description

The 8-bit processor REIGHT 1 was developed with a subset of the AVR® instructions but offers all the functions to run full-fledged microcontroller programs.

With its two staged pipeline it is capable of running at 130Mhz clock speed. In order to show the structure of the processor, at first the selected set of instructions is shown below.

1.1. Instruction set

Instruction	operand	description	operation	FLAGS	clock cycles
ADD	R_d, R_r	add without carry	$R_d \leftarrow R_d + R_r$	Z,C,N,V,S	1
ADC	R_d, R_r	add with carry	$R_d \leftarrow R_d + R_r + C$	Z,C,N,V,S	1
SUB	R_d, R_r	subtract without carry	$R_d \leftarrow R_d - R_r$	Z,C,N,V,S	1
SUBI	R_d, K	subtract immediate	$R_d \leftarrow R_d - K$	Z,C,N,V,S	1
AND	R_d, R_r	logical and	$R_d \leftarrow R_d \& R_r$	Z,N,V,S	1
ANDI	R_d, K	logical and with immediate	$R_d \leftarrow R_d \& K$	Z,N,V,S	1
OR	R_d, R_r	logical or	$R_d \leftarrow R_d \mid R_r$	Z,N,V,S	1
ORI	R_d, K	logical or with immediate	$R_d \leftarrow R_d \mid K$	Z,N,V,S	1
EOR	R_d, R_r	exclusive or	$R_d \leftarrow R_d \wedge R_r$	Z,N,V,S	1
COM	R_d	one´s complement	$R_d \leftarrow \$FF - R_d$	Z,C,N,V,S	1
SBR	R_d, K	set bit(s) in register	$R_d \leftarrow R_d \mid K$	Z,N,V,S	1
INC	R_d	increment	$R_d \leftarrow R_d + 1$	Z,N,V,S	1

Instruction	operand	description	operation	FLAGS	clock cycles
DEC	R_d	decrement	$R_d \leftarrow R_d - 1$	Z,N,V,S	1
TST	R_d	test for zero or minus	$R_d \leftarrow R_d \& R_d$	Z,N,V,S	1
CLR	R_d	clear register	$R_d \leftarrow R_d \wedge R_d$	Z,N,V,S	1
SER	R_d	set register	$R_d \leftarrow \$FF$	NONE	1
branch instructions					
RJMP	K	relative jump	PC ← PC + K + 1	NONE	1
RCALL	K	relative call subroutine	PC ← PC + k + 1	NONE	2
RET		subroutine return	PC ← Stack	None	4
CP	R_d, R_r	compare	$R_d - R_r$	Z,C,N,V,S	1
CPI	R_d, K	compare with immediate	R_d- K	Z,C,N,V,S	1
BRBS	s, K	branch if status flag set	If (SREG(s) = 1) then PC←PC+K+1	NONE	1/2[2]
BRBC	s, K	branch if status flag cleared	If (SREG(s) = 0) then PC←PC+K+1	NONE	1/2
BREQ	K	branch if equal	If (Z = 1) then PC←PC+K+1	NONE	1/2
BRNE	K	branch if not equal	If (Z = 0) then PC←PC+K+1	NONE	1/2
BRCS	K	branch if carry set	If (C = 1) then PC←PC+K+1	NONE	1/2
BRCC	K	branch if carry cleared	If (C = 0) then PC←PC+K+1	NONE	1/2

[2] taken / Not taken

Instruction	operand	description	operation	FLAGS	clock cycles
BRSH	K	branch if same or higher	If (C = 0) then PC←PC+K+1	NONE	1/2
BRLO	K	branch if lower	If (C = 1) then PC←PC+K+1	NONE	1/2
BRMI	K	branch if minus	If (N = 1) then PC←PC+K+1	NONE	1/2
BRPL	K	branch if plus	If (N = 0) then PC←PC+K+1	NONE	1/2
BRGE	K	branch if greater or equal, signed	If (S = 0) then PC←PC+K+1	NONE	1/2
BRLT	K	branch if less than, signed	If (S = 1) then PC←PC+K+1	NONE	1/2
BRVS	K	branch if overflow flag is set	If (V = 1) then PC←PC+K+1	NONE	1/2
BRVC	K	branch if overflow flag is cleared	If (V = 0) then PC←PC+K+1	NONE	1/2
data transfer instructions					
MOV	R_d, R_r	copy register	R_d ← R_d	NONE	1
LDI	R_d, K	load immediate	R_d ← K	NONE	1
LD	R_d	load indirect	R_d ← (Z)	NONE	1
ST	R_d	store indirect	(Z) ← R_d	NONE	1
PUSH	R_d	push register on stack	$_{STACK}$ ← R_d	NONE	1
POP	R_d	pop register from stack	R_d ← STACK	NONE	1

Instruction	operand	description	operation	FLAGS	clock cycles
bit and bit-test instructions					
LSL	R_d	logical shift left	$R_d(n+1) \leftarrow R_d(n)$ $R_d(0) \leftarrow 0$, $C \leftarrow R_d(7)$	Z,C,N,V	1
LSR	R_d	logical shift right	$R_d(n) \leftarrow R_d(n+1)$ $R_d(7) \leftarrow 0$, $C \leftarrow R_d(0)$	Z,C,N,V	1
ROL	R_d	rotate left through carry	$R_d(0) \leftarrow C$ $R_d(n+1) \leftarrow R_d(n)$ $C \leftarrow R_d(0)$	Z,C,N,V	1
ASR	R_d		$R_d(n) \leftarrow R_d(n+1)$	Z,C,N,V	1
SEC		set carry	$C \leftarrow 1$	C	1
CLC		clear carry	$C \leftarrow 0$	C	1
NOP		no operation		NONE	1

Table 1: instruction set

1.2. Structure

A processor functions by executing a program composed of sequential instructions. These instructions reside in the program memory (PM), where each address corresponds to a single instruction. Address 0, consequently, marks the inception of the program. The REIGHT 1 accommodates a program memory capable of holding 512 16-bit operations, denoting the maximum executable program size for this processor.

The program counter (PC), situated ahead of the PM, increments the memory address by one per clock cycle with exceptions clarified later. The resulting address is fed into the PM, generating the 16-bit instruction output.

This instruction, known as an opcode, is fed threw a pipeline stage, detailed subsequently, before reaching the decoder. The decoder translates this instruction into control signals, determining actions across the system, including addresses for the register file (RF) and operation codes for the arithmetic-logic-unit (ALU).

The RF, a small volatile storage comprising 32 bytes, contains variables instrumental in ongoing processor operations. Notably, RF Address 30 and 31 are used for the random-access-memory (RAM), jointly forming a 10-bit address termed Z. The decoder yields two read addresses, converted by the RF into outputs. Additionally, a write address is supplied by the decoder to write data into the RF. This address is routed through pipeline stage (PS) two before going into the Register File.

Subsequently, the outputs of RF Operand A (OPA) and Operand B (OPB) progress through PS 2 to reach the ALU. This unit executes logical operations—addition, subtraction, AND, OR, XOR, and shifts. The ensuing results are written to their designated writing address in the RF. Simultaneously, the ALU updates flags within the status register (SREG) crucial for decision-making in the program. Furthermore, the decoder can trigger an ALU operation without enabling RF write enable (WE), altering the SREG without affecting the RF—a feature employed for comparisons.

The ALU also accommodates operations with a constant instead of OPB, sourced directly from the instruction. This capability extends to loading values from the program into the RF by channeling the constant through the ALU and updating the RF.

In tandem with the ALU, the data memory (DM)—the processor's RAM—extends the capabilities of the RF. The RF addresses 30:31 (Z) serve as the 10-bit RAM address. The

decoder can activate a WE signal to store OPA within the specified Z address in the DM. This is triggered by the instruction ST. An alternative method entails altering the Z address via a multiplexer (MUX) to the stack address. This stack address is determined by the stack pointer (SP), initially set at $3FF (1023), denoting the RAM's uppermost address. During a PUSH operation (store on stack), OPA is stored at the stack address, followed by a decrement of the SP value by 1. To retrieve data from RAM, a POP or LD (Load) operation is executed. In the LD operation, the value of RAM address Z is retrieved and swapped with the ALU's result signal using a MUX. POP decreases the SP value and uses the stack address to access the data. The instructions ST (store), LD (Load), PUSH and POP are completed within a single cycle on the REIGHT 1, a special feature detailed later.
Certain RAM cells serve dual purposes, serving as inputs and outputs for the REIGHT 1. These cells, mirrored as registers in the design, permit parallel utilization by external sources alongside the RAM, which operates with DRAM, outputting one byte at a time.

Within the instructions some modify the PC—RJMP, the simplest, adds a constant value to the current PC value. Further insights into PC functionalities are explored in the special features section. A more intricate instruction, the branch, serves as a conditional jump instruction. When the condition (a comparison to a flag in SRAM) is fulfilled, the jump is executed; otherwise, the PC increments by 1 as usual. The most complex instruction pair, RCALL and RET, facilitates function calling and return within the program. This necessitates storing the old PC value in the stack during the call and substituting it with a new value. For the return, the old PC value is retrieved from the stack and reinstated into the PC.

1.2.1 Pipelines

Within logical circuits, a significant challenge arises with signal transit times—the duration it takes for a signal to traverse from one point to another on the PCB. These delays stem from the individual transistors' switching times, accumulating throughout the processor's circuitry. The sequence is such that only when the preceding element switches the subsequent follows suit. In complex processors, these signal transit times can extend, especially evident in the journey from the program counter to the ALU, often surpassing a single clock cycle. To counter this and achieve higher clock frequencies, the implementation of pipeline stages becomes crucial.

During the implementation of the pipeline, the processor is divided into several parts; in case of REIGHT 1, three. This division is based upon the assumption that while some sections of the processor are engaged in switching others must wait their turn. By dividing the processor the sections can work on a problem without having to wait their turn.

These three integral parts of the REIGHT 1 are:

1. instruction fetch: This encompasses the PC and the PM. In this stage the new instruction to process is being fetched.
2. decode stage: This stage involves the decoder, the RF, and some forwarding MUXs, which will be explained later. In this stage, the decoding of the instruction and the setting of control signals happens, as well as the storing and reading the RF.
3. execute Stage: This stage includes crucial components like the ALU, the DM with SP, Ports, and the SREG. This is where the calculation, but also the storing and loading of the DM takes place.

Each of these stages is divided by a pipeline stage, which functions as a clocked register routing all signals from the preceding section into the subsequent one. This mechanism significantly truncates the signal transit times to the duration of a single stage. However, it also implies that an instruction takes tree clock cycles to execute. This scenario gives rise to two critical issues:

1. operational dependency: When an instruction's operand relies on the previous instruction's operand, the result is not yet stored in the RF. To resolve this,

Forwarding MUXs are employed, facilitating the exchange of the respective operand with the output of the execute stage. Proper control of these MUXs demands comparison between the previous write address and the current read address of the RF. When the addresses match and the previous WE signal is active, the signal for the specific operand must be forwarded.

2. branch calls: There is a possibility that when a branch is invoked, the corresponding status flag is not set yet (due to the clocked SREG). The most straightforward approach to ensure a correct branch call involves pausing the PC momentarily, awaiting one clock cycle, and then deciding whether to take the branch or not. The REIGHT 1 manages to execute a Branch in one clock cycle (if taken) and two cycles (if not taken), a special feature detailed further in the corresponding chapter.

The described REIGHT 1 is shown as a block diagram in the following subchapter.

1.2.1 Block diagram

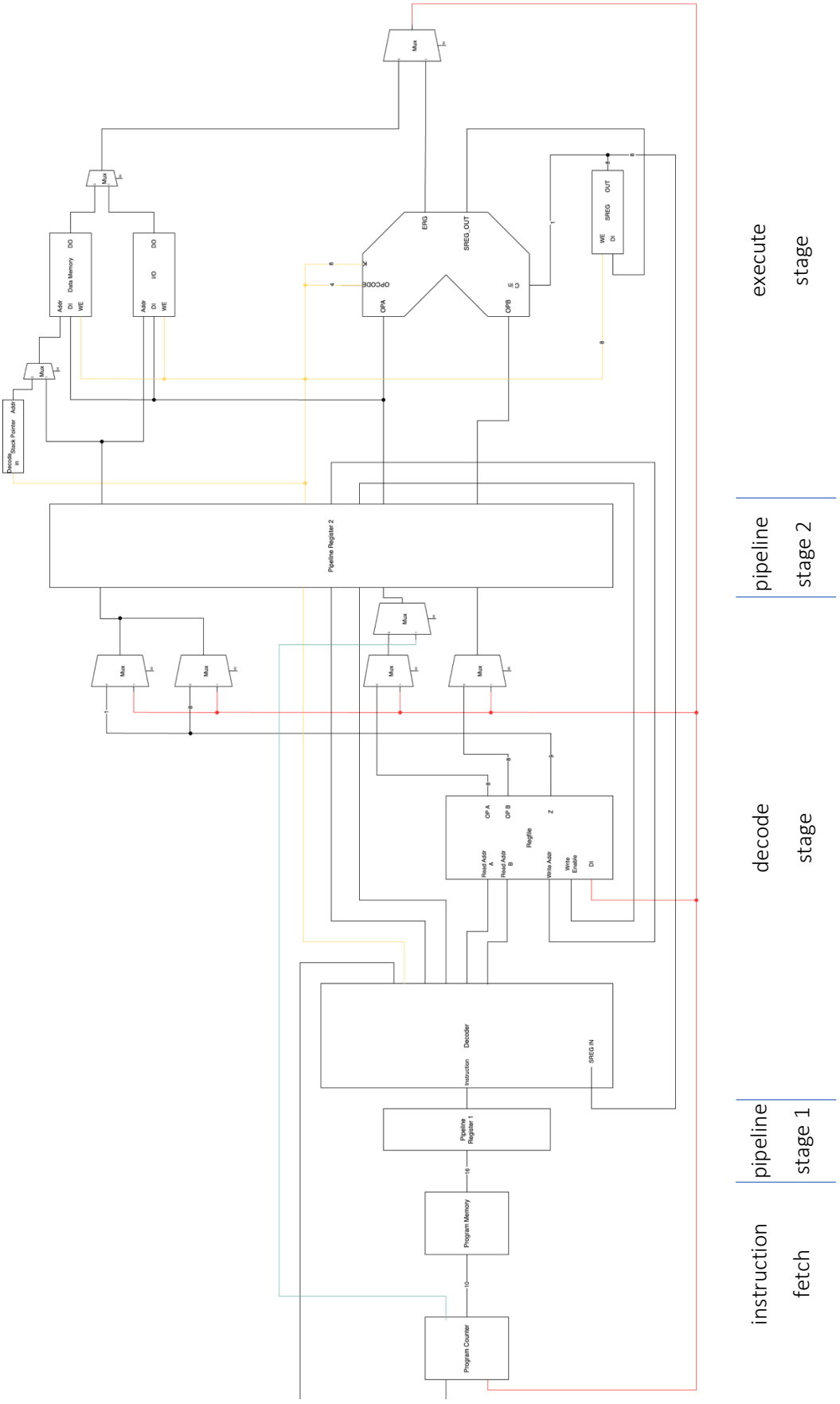

Figure 1: block diagram top level

2. Special Features

This processor utilizes part of the AVR® instruction set. To emphasize its special features, a comparison with AVR® processors is evident. The most crucial point of comparison is performance. However, since some instructions are multi-cycle operations, relying solely on clock speed is insufficient to describe the processor's performance accurately. Hence, the instruction per cycle (IPC) index is employed. This IPC represents the average duration of an instruction, considering the frequency of occurrence. The smaller the IPC, the faster the processor operates. Consequently, efforts should focus on optimizing multi-cycle operations to minimize the number of cycles. All optimizations aimed at enhancing the speed of multi-cycle operations are categorized as special features. To delve deeper into these features, the multi-clock cycle instructions of the AVR Instruction set are compared to the same implemented instructions of REIGHT 1, specifically examining the required cycles for each instruction.

instruction	used cycles AVR®	used cycles REIGHT 1
Push	2	1
Pop	2	1
LD	2	1
ST	2	1
Branch(taken)	1	1
Branch(not taken)	2	2
RJMP	2	1
RCALL	3	2
RET	4	4

Table 2: comparison of the used cycles AVR® and RIGHT 1

To achieve these speed improvements, optimizations were implemented on the PC, SP, and the RCALL/RET Process. These three optimizations constitute the main special features of this processor and are explained in more detail in the following subsections.

2.1. Program counter

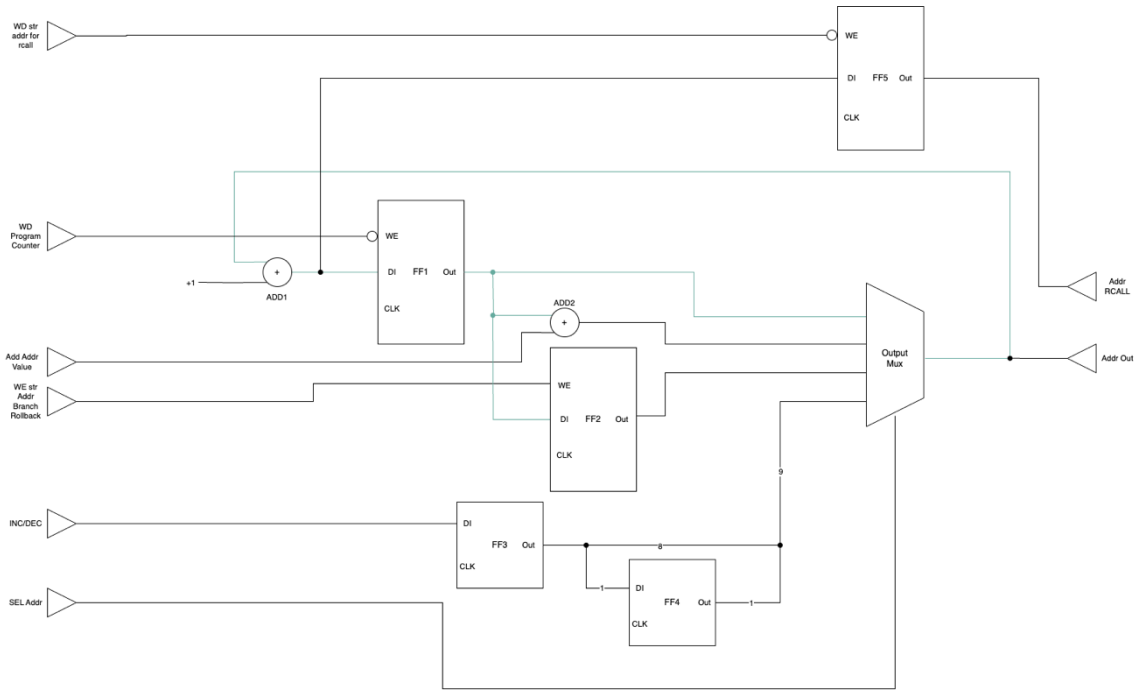

Figure 2: block diagram Program Counter

Figure 2 describes the structure of the PC. The PC is a special feature of the REIGHT 1 because it enables the processor to execute instructions that are multi-cycle in the AVR® instruction set with fewer cycles. To delve deeper into its functionality, it is necessary to explore the various operation modes of the PC, with particular focus on the operations to be performed in relation to the clock and how these operations work.

The Program counter needs to realize the following tasks:

- normal operation:
  - This operation describes all instructions that do not make changes to the PC. In this case, the PC value increments by one count per clock cycle.
  - In Figure 2, this normal operation is realized by ADD1, which increments the PC value by one, and FF1, which stores the PC value. Every clock cycle, FF1 updates the PC value with the output of ADD1. The output MUX is selecting port 0 to give the FF1 signal out.
- RJMPS:
  - This involves adding a value to the PC.

- To realize this feature, the ADD2 is adding the jump value stored in the instruction to the PC value, and output MUX is selecting port 1. This jump value is 9-bit (-255 ≤ k ≤ +255).

- branches:
  - Branches modify the PC value when a condition is met. This change on the PC value is equivalent to a RJMP with a 7-bit jump (-64 ≤ k ≤ +63).
  - Since the processor utilizes a pipeline, at the time of decision whether a branch is taken or not, the decision condition is not necessarily set. It is generated in the execute stage and is available in the subsequent clock cycle. However, as it is more likely that the branch will be taken than not, the branch is executed on suspicion, and a jump value is added to the PC. Simultaneously, the old PC value is stored. If it turns out in the next clock cycle that the branch should not have been taken, the old PC value must be restored and all WE signals from the decoder are deactivated, thus making the current instruction in the decoder invalid.
  - In case of a branch, the FF2 gets a WE signal from the decoder to save the old PC value. Meanwhile, ADD2 is adding the jump value to the PC value.
  - The output MUX is selecting port 1 to give the ADD2 Signal out, and the FF1 will accept the added jump value +1 on the next clock.
  - In the following clock cycle, if it becomes obvious that the branch is taken correctly, nothing else will happen in that regard.
  - If the branch is taken falsely, a PC value rollback must happen. This is done by the output MUX, which selects port 2 to give through the saved PC value.

- RCALL:
  - An RCALL is a jump to a function where the previous PC value is stored in the stack. Since the stack is only one byte per memory cell in size and the PC is 9 bits long, storing it must be done with two individual pushes onto the stack. For this reason, the current PC value is stored in a register at the beginning of the RCALL process. Meanwhile, the PC is paused, and the lower 8-bit of PC value is pushed into the stack. The second push occurs

in the 2nd cycle of the RCALL, along with the addition of a constant to the PC value.

- In Figure 2, FF5 is saving the ADD1 output value on every cycle. By giving a write disable (WD) signal, this storing is paused, and the current PC value remains in FF5. Also, the FF1 and the PR 1 get a WD signal, ensuring that on the next clock cycle, no new instruction is fed into the decoder. In order to distinguish the first cycle of the RCALL from the second, a state machine is started which saves the current state.
- On the first clock cycle of RCALL, the decoder exchanges the OPA value with the lower 8 bits of the PC value stored in FF5. This change happens in a MUX seen in Figure 1 (turquoise signal). The decoder also sets the signal for a push onto the stack — Because of the pipeline structure, this push will happen on the next clock cycle.
- While the first push is executed in the execute stage, the decoder exchanges the OPA value to the upper bit, as bit 0 of a byte, of the PC value stored in FF5 and fills the remaining bits with zeros. The decoder also sets the signals for pushing a byte into the stack.
- In the second cycle of the RCALL instruction, the jump value is added to the PC value by ADD2. The PC output MUX is selecting port 1 and the WD signals are reset. The state machine is reset, and at this point, the processor can continue to run the program normally.

- RET:
  - In case of RET, the PC value stored in the stack is read and loaded into the PC. As this value is 9 bits in length, the most upper bit is read first and stored in a shift register (FF3 and FF4); this register is running constantly and gets the execute out as input. second, the lower 8 bits are loaded from the stack and shifted into the shift register. Afterward, the temporarily stored PC value is loaded into the PC.
  - This operation needs to be 4 Clock cycles long because of the Pipeline structure of the Processor.

- When a RET instruction reaches the decoder, firstly the PC is paused by setting the WD for the FF1. Secondly, the PR 1 is getting a WD signal as well. This ensures that on the next clock cycle, no new instruction is fed into the decoder. The decoder now uses the state machine again, which is initially at state 0. In order to dive deeper into the behavior, the states of the state machine are explained below.
  - State 0:
    - The decoder sets the signals for a POP.
  - State 1:
    - The decoder sets the signals for a POP again.
    - In the execute stage, the first POP is executed, and the FF3 in Figure 2 saves the value of it at the beginning of the next cycle.
  - State 2:
    - The second POP is executed. And the FF3 saves the value of it at the beginning of the next cycle while the lower bit of the old FF3 is shifted into FF4; this bit is the upper bit of the PC value to return to.
  - Stage 3:
    - The decoder resets the state machine.
    - The decoder sets the select signal of the PC output MUX to port 3, which loads the saved PC value. This value resembles the address of the RCALL + 1.

2.2. Stack pointer

The SP needs to have two main functions. First, a post-decrement of the SP value in case of a PUSH. This means the storing of a value at the stack address in the DM takes place, and afterward, the stack value is decreased by one, indicating the next free address for a PUSH. Second, a pre-increment of the SP value in the case of a POP. In this case, the SP value must be incremented before the load of a value in the SM at the stack address takes place. To gain a better understanding of the designed SP, Figure 3 is shown.

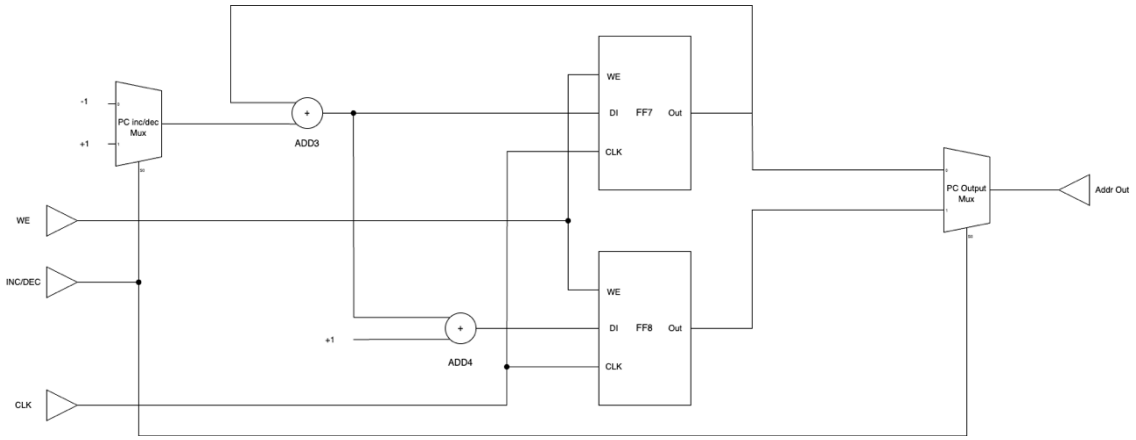

Figure 3: block diagram Stack Pointer

Implementing a post-decrement within a single clock cycle is relatively straightforward utilizing the output of FF7. To ensure the post-decrement, the PC inc/dec MUX selects -1, while the decoder is setting a WE signal for FF7. Consequently, at the start of the subsequent cycle, FF7 decrements by 1. However, executing a pre-increment presents a challenge due to FF7 updating only at the next cycle's onset. This limitation restricts achieving a pre-increment and load within a single cycle while solely relying on FF7.

To resolve this, the output of ADD3 could be used to create the output address of the SP in case of a pre increase. In that case, a PC output MUX is used which switches the FF7 output to the ADD3 output. This makes a pre-increment and a load in one cycle possible. However, this solution introduces challenges, primarily due to the extended signal pathways from the SP. Moreover, the use of an adder like ADD3 with comparatively long switching times, in relation to the length of a clock cycle, rekindles concerns regarding excessive signal transit times.

Consequently, achieving the targeted 130MHz clock speed for the REIGHT 1 becomes unfeasible when solely relying on ADD3 for a post increment. To overcome this hurdle,

REIGHT 1 incorporates a second flip flop (FF), FF8, which stores the current SP value incremented by 1. FF8 holds this incremented value ready in the event of a POP operation. Meanwhile, the actual Stack pointer, residing in FF7, undergoes a post-increment. The PC output MUX seamlessly toggles between FF7 and FF8 outputs, based on whether the instruction is a PUSH or POP.

2.3. Data memory

The Data Memory is sleekly designed, so LD(Z) or ST(Z) is executed in 1 clock cycle without having problems with the signal transit times. This is probably due to the fact that the RAM used in the REIGHT 1 is faster than that of the AVR® processors and thus enables one-clock execution. However, this is just a guess, the exact reason why AVR® processors need 2 cycles for these instructions is not known to the creator of the REIGHT 1.

3. Performance evaluation

Relying solely on a processor's clock speed doesn't paint the full picture of its performance. The complexity arises from the processor's capability to execute multiple clock instructions. Thus, assuming that a high clock-speed processor, despite handling more clock cycles for multiple instructions, outperforms a slower clock-speed processor with fewer clock counts for these instructions is misleading. This realization led to the inclusion of the instruction per cycle (IPC) index in addition to the clock speed in performance evaluations.

The IPC index factors in the efficiency of instruction execution. It involves multiplying the probability of an instruction occurring by its cycle count and summing these calculations. This resultant index represents the average instruction executed within a clock cycle.

In crafting the IPC, a specific instruction mix is employed: 15% load, 5% store, 16% branch (taken), 4% branch (not taken), and 60% arithmetic/logical operations.

The computation of the IPC is detailed below.

Instruction type	probability of occurring in %	used clock cycles
LD	15	1
ST	5	1
Branch (taken)	16	1
Branch (not taken)	4	2
Arithmetic/logical	60	1
	IPC:	1,04

Table 3: calculation of IPC

This IPC is implementable with a clock frequency of 130 MHz[3]. This outstanding performance is possible while only consuming 99mW of Power.

The Artix-7™ FPGA[4] on the Basys 3 board is utilized by the following amount.

Resource	Utilization	Available	Utilization in %
LUT	780	20800	3,75
LUTRAM	128	9600	1,33
FF	572	41600	1,38
IO	50	106	57,17
BUFG	1	32	3,13

Table 4: utilization of the Basys 3 board

[3] worst negative slack (Setup Timing) of 0,214 ns
[4] Part: xc7a35tcpg236-1

4. Possible performance gains

Performance could be gained in 3 ways:

1. increasing the clock speed
2. lowering the IPC
3. make crucial changes to the architecture of the processor.

4.1. Increasing the clock speed

By increasing the clock speed, the slowest link in the processor becomes a problem in the signal transit times. In the case of the RIGHT 1, this slowest link is the ALU. To improve signal transit times in the execute stage, a third pipeline stage must be realized. This PS would be placed after the ALU and would buffer not only the output of the ALU but also the DM and the I/O.

This would require changing the handling of some instructions. A secondary forwarding MUX set must be installed after the PS 2 to ensure that the ALU, I/O, and the DM can utilize results of the earlier clock cycle. The branch instruction could function as if there were no third pipeline register because the SREG is not fed through the PR3. But other instructions such as RCALL and RET would be slowed down significantly. It is also questionable whether a third pipeline register will bring the desired improvements because the instruction fetch and the decode stage are both very close to the maximum in terms of timing. To crank up the clock speed even more, more pipelines need to be installed in the processor design. This would push up the IPC and would slow down the processor in that way.

4.2. Lowering the IPC

A perfect IPC, with this kind of processor architecture, would be a value of 1. to achieve this IPC the branch not taken cycle counts must be lowered from two to one. This would bring relatively more complexity to the design. In order to keep the pipeline structure of the processor, there are two ways to lower the clock count.

1. Not every branch condition is set in the previous clock cycle. By checking if the previous operation had a WE signal to the relevant flag in the SREG register, the branch decision could be made in the cycle of the branch call in the decoder stage.

This would lower the branch clock count to one if the branch condition is set on an earlier clock than the previous one. Nevertheless, this was tried in the development of the REIGHT 1 and brought significant problems with signal transit times. Furthermore, it should be considered that this case isn't the norm, and the branch condition is set most likely in the previous cycle.

2. Doubling both the instruction fetch and decode stages. In this approach, the initial IF and Decode pairing would generate control signals for the subsequent instruction in the event of a branch being taken, while the second pairing would handle control signals if the branch were not taken. To accommodate the branch condition, alterations to the input signals of PS 2 could be implemented. However, it's essential to note that this adjustment signifies a substantial architectural shift, albeit resulting in a mere 0.04 improvement in the IPC. If contemplating such significant modifications, it might be prudent to explore additional architectural enhancements described in section 4.4. Crucial changes to the architecture of the processor, which could substantially elevate the processor's overall speed and performance.

4.3. Summary of the previous suggestions for improvement

The previous two subchapters indicate that potential performance improvements with maintaining the current architecture of the processor are very minimal. Hence, the REIGHT 1, with its current architecture, is close to an optimum level of achievable performance. In comparison to similarly positioned microcontrollers in the market, the REIGHT 1 positions itself in the high-performance areas (one-core processors). For instance, when compared to the ATMega 328P, a microcontroller commonly used in the amateur field (also utilizing AVR® instruction set), the REIGHT 1 is more than 8 times faster. (ATMEL, 2015)

However, if the REIGHT 1 intends to target a different market that demands significantly higher performance, substantial architectural changes will be necessary. These changes are outlined in Chapter 4.4. Crucial changes to the architecture of the processor.

4.4. Crucial changes to the architecture of the processor

To gain significantly higher performance, a second processor core could be added. This would halve the IPC (Instructions Per Cycle) but also require more complex programming of the microcontroller. Additionally, a cache could be implemented in the processor to expedite RAM loading times and facilitate easy data exchange between both cores.

Furthermore, the processor could have multiple ALUs (Arithmetic Logic Units) that could be utilized to execute instructions speculatively. The instruction fetch (IF) stage could retrieve multiple instructions simultaneously, which would be processed in parallel during the execute stage.

Another performance improvement would involve adding a branch cache. This cache stores the latest branch jumps and could predict upcoming branches before the decoder. However, these changes are very complex but could elevate the design to a whole new level.

5. Included software

The REIGHT 1 is delivered in a bundle with an 8-bit calculator software. This software is utilizing the Power of the REIGHT 1.

5.1. Key Features

- **Effortless Arithmetic:** Seamlessly add, subtract, multiply, and divide with precision using our 8-bit arithmetic operations.

- **Versatile Calculations:** Perform 8-bit addition and subtraction with ease, accommodating positive operands and delivering precise 9-bit results, whether positive or negative.

- **Powerful Multiplication:** Unlock the potential of 8-bit multiplication with positive operands, generating expansive 16-bit outcomes for complex computations.

- **Precise Division:** Experience accurate 8-bit division with positive operands, obtaining concise integer results for your mathematical needs.

- **Visual Feedback:** Witness your results come alive! See your result as a detailed 16-bit binary number, vividly displayed across 16 LEDs on the REIGHT 1, starting from the most significant bit on the left.

- **Intuitive Display:** Effortlessly view results on the 7-segment display, showcasing numbers from 9999 to -255. No worries if your results exceed these limits—the display automatically resets and the 16 bit output LEDs will have your back.

- **Simplicity Redefined:** Easily input operands using the 16 switches on the REIGHT 1 board. The first 8 switches represent operand A, while the remaining 8 denote operand B, simplifying your data entry process.

- **Seamless Operation Selection:** Swiftly choose your preferred mathematical operation using the intuitive built-in buttons, enabling smooth transitions between calculations.

button	operation
left	addition
up	division
right	multiplication
down	subtraction
middle	reset of 7-segment display

6. Bibliography

ATMEL. (2015). *ATmega328P Datasheet.* Retrieved from https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Digital attachment

1. VHDL description of processor and Vivado project: /Vivado/Prozessor
2. included calculator software: /Taschenrechner.asm

Declaration of independence

I, Ruben Hermann Felix Saitz, hereby declare that I have independently prepared the present work using only the specified sources and aids. This work has not been submitted to any examination authority in the same or similar form and has not been published previously.

Erfurt, January the 2nd of 2024
place, date


Signature